A Block Diagram Compiler

By JOHN L. KELLY, Jr., CAROL LOCHBAUM, and V. A. VYSSOTSKY

(Manuscript received December 7, 1960)

A computer program known as BLODI, which accepts for an input a source program written in the BLODI language, is described. The BLODI source language corresponds closely to an engineer's block diagram of a circuit and is easily learned, even by persons not familiar with computing machines. The input code consists essentially of designating the connectivity of a number of boxes drawn from an alphabet of about 30 types. These types include amplifiers, delay lines, counters, etc., which are familiar to designers of electronic circuits. The principles of the compiler are explained and applications are discussed.

I. INTRODUCTION

This paper describes a computer program known as BLODI (BLOck DIagram compiler). BLODI accepts for an input a source program written in the BLODI language, which corresponds closely to an engineer's block diagram of a circuit, and produces a machine program to simulate the circuit. BLODI has been written for both the IBM 704 and 7090 machines, and has been in use at Bell Telephone Laboratories for several months. Generally speaking, there are two situations in which it can be used profitably. One arises when a person with no knowledge of machine coding wishes to program his own problem. In this case, the BLODI language is much easier to learn than Fortran or SAP. There are, in addition, certain problems involving a rather smooth flow of data which can be most easily coded in BLODI, even by an experienced programmer.

It is rather easy to estimate the efficiency of an object program produced by BLODI. Thus a person with no knowledge of computing machines can often tell if he should code his problem in BLODI or seek the aid of an experienced programmer. This will be discussed in Section V.

BLODI was written to lighten the programming burden in problems concerning the simulation of signal-processing devices. It has the added advantage of keeping the engineer who invents such a device in close communication with the computing machine by eliminating the middleman (expert programmer).

II. BLOCK DIAGRAM OF SAMPLED (OR PULSE) SYSTEMS

The circuits* which we wish to consider here are limited to combinations of devices which accept pulses as inputs and yield pulses as outputs. While the pulses may have arbitrary sizes within certain limits, they must all occur at multiples of a fixed clock time. In general, the output of one of the devices (or boxes) can depend on the present and all past input pulses. A box whose output is independent of the current input pulse or pulses is called a *delaying-type box*. In the current form of the compiler the only delaying-type box is a simple delay line. In addition to these boxes, the circuit may have one or more ultimate outputs and original inputs. A *circuit* then means a number of boxes, ultimate outputs, and original inputs connected in such a way that the output of any box is connected to one or more inputs to boxes and ultimate outputs, and each input to a box is connected to an output from a box or an original input. (We limit ourselves to boxes with a single output.)

A *closed loop* is a path that starts from any point, goes only through connected boxes in the direction of the pulses (i.e., input to output), and returns to its starting point. A circuit which does not contain a closed loop with no delaying-type boxes will be called an *admissible* circuit. The compiler will reject any block diagram which does not describe an admissible circuit. It is easy to see that if the pulse heights were limited to a finite number of values (as they are, of course, in the machine simulation) then an admissible circuit would be a finite-state machine. On the other hand, there is no way to interpret a block diagram corresponding to a nonadmissible circuit. To be sure, one could connect physical boxes in such a manner and something would happen. The analysis, however, would involve the precise transient behavior of the devices within the pulse width — information which is not available to the compiler.

In addition to simulating pulse circuits, the compiler may be used to simulate continuous circuits whose inputs and outputs are bandlimited time functions. One must first design a pulse circuit whose output pulses would correspond to the sample values of the desired output. Extreme

---

* For clarity the machine being simulated will be called the *circuit*. Its description in a certain canonical form will be called the *block diagram*. The word *machine* will always mean the IBM 704 EDPS (or 7090 EDPS).

care must be exercised here; for example, an accumulator (a device whose output is the sum of all previous input pulses) is not equivalent to an integrator. Certain continuous circuits (especially nonlinear ones) are extremely difficult to translate into pulse form. A simple circuit which is difficult to simulate on a machine (with or without the use of this compiler) is the following: Let a bandlimited input signal be connected to a full-wave rectifier, and this to a low-pass filter to return the signal to the original bandwidth.

We will see later that (with a trivial exception) the compiler can be used to simulate any admissible circuit composed of boxes drawn from a fixed list or stockpile. The boxes may have only one output (which may go to several places, however) and at most four inputs. The first constraint is no real restriction, but the second one is.* We know of no example in signal processing where a general function of more than four variables that cannot be expressed as combinations of functions of four or less variables is needed. In fact, two inputs to each box would probably be adequate but slightly awkward.

III. THE BLODI LANGUAGE

A BLODI source program is punched on standard SHARE symbolic cards in either the FAP (7090) or SAP (704) format. In general, each card corresponds to one box in the circuit; there is, however, a provision made for continuing a description of a box to the next card. The location field (columns 1 through 6) is either blank or contains the name assigned the box by the programmer. (If a box is to have any inputs, it must have a name.) The operation code field (columns 8 through 10) contains the type of box. Parameters (such as gain of an amplifier) and output connections are separated by commas and listed consecutively starting in column 12 (or column 16 for the 7090 format). The various inputs to the same box are designated by a fraction bar and numeral following the name of the box. Example:

```
UV AMP 5.28, XY, Z/2
```

Box UV is an amplifier with a gain of 5.28 which feeds box XY (first input) and the second input of box Z.

A list of all the available box types appears in Table I. Note that INP may be thought of as a box which generates signals spontaneously; actually it obtains its input from a tape designated as a parameter. Simi-

---

* Technically speaking, this is not true. A circuit could be designed corresponding to any finite state machine. However, we consider that it is not in the proper spirit to take advantage of the fact that the pulse heights are limited to 2^{36} values.

TABLE I — ALL THE TYPES OF BOXES WHICH ARE RECOGNIZED BY THE COMPILER

Type	Function	Inputs	Parameters
DEL	Delay	Signal	Number of units delay
AMP	Amplifier	Signal	Gain
ADR	Adder	1–4 Signals	None
SUB	Subtractor	+ Input − Input	None
MAX	Maximum circuit	1–4 Signals	None
MIN	Minimum circuit	1–4 Signals	None
CLP	Positive clipper	Signal	Clipping level
CLN	Negative clipper	Signal	Clipping level
SCL	Symmetric clipper	Signal	Clipping level
FWR	Full-wave rectifier	Signal	None
BAT	Battery or bias	(Signal)	Bias
MRP	Multiplier	2 Signals	None
DIV	Divider	Dividend Divisor	None
SQT	Square rooter	Signal	None
ACC	Accumulator	Signal	Gain
FLT	Transversal filter	Signal	Number of taps Delay per tap Gains
SLF	Symmetric filter	Signal	Number of taps Delay per tap Gains
AFL	Antisymmetric filter	Signal	Number of taps Delay per tap Gains
QNT	Quantizer	Signal	Number of levels Levels
LQT	Linear quantizer	Signal	Step size
SMP	Sampler	Signal	Period Quiescent level Initial phase
HLD	Sample and hold	Signal; control	Threshold
CNT	Counter	Signal	Countdown factor Threshold Active level Passive level Initial phase
DTS	Double-throw switch	Control; 2 signals	Threshold
FLF	Flip-flop	Signal	Low threshold High threshold Low state output High state output
PLS	Pulser	Control	Threshold Pulse length Pulse level Quiescent level
CØS	Cosine generator	None	Period Phase Amplitude
GEN	Function generator	None	Period Sample values
WNG	Noise generator	None	Standard deviation
PRT	Printer	Control; 3 signals	Threshold Record Limit

Table I — (*Continued*)

Type	Function	Inputs	Parameters
INP	Input	None	Tape number File maximum Samples per record Record maximum Start printing Stop printing
ØUT	Output	Signal	Tape number File maximum Samples per record Record maximum Start printing Stop printing
END	Last card of source program		

larly, OUT causes the signal appearing on its input lead to be written on the designated tape. A circuit may have several inputs and outputs. END is not a box at all but signifies the end of the source program. In addition to the types listed, it is possible for a programmer to create types of his own invention by supplying subroutines written in basic machine language. It is also quite easy to change the basic input-output programs used by BLODI to handle arbitrary tape formats.

IV. PRINCIPLE OF OPERATION

An object program produced by the compiler consists of three parts:

(a) the *prefix*, which sets up the logic for the main loop;

(b) the *main loop*, which is executed once for each sample processed;

(c) the *suffix*, which causes the main loop to be repeated the proper number of times, empties output buffers, fills input buffers, etc.

We will concern ourselves here only with the main loop. Except for some strictly local inner loops in certain boxes, this part of the object program is compiled in the same order in which it is to be executed. Simply stated, the procedure is as follows: One storage cell in the object program is assigned for each box. Each time the main loop is entered, these cells will contain values corresponding to the last *outputs* of the respective boxes. It is then the function of the main loop to compute these output values for the next (current) time slot and to fill the cells with these values.

In order to simplify the description of the algorithm used by the compiler we will at first limit ourselves to the case where all delays are unit delays. By "compiling a box" we mean writing the necessary coding to cause the object program to fill the output cell of the box with the current pulse value. A nondelaying-type box cannot be compiled until all the boxes feeding it have been compiled. The reason is that the output of this type of box is a function of its current inputs, and this part of the object program must not be executed until the cells corresponding to input to this box have been filled with current pulse values. On the other hand, a unit delay must be compiled *before* the box which feeds it. Its output is a function of (equal to, in fact) its *last* or *old* input. At object time this value must be "moved along" before it is overwritten. To reword this second rule, no box which feeds a delay line can be compiled until that delay line has been. This second rule could be dropped if we provided an additional storage cell for each unit delay and had the object program first go through and fill each of these cells with the old input to the corresponding delay line. We will see below, however, that the only price we pay for the more efficient procedure is that the compiler will reject any block diagram containing a closed loop with nothing but delays. Such a diagram would represent an admissible circuit but would be of little value, since we could never get anything but zeros out of this loop at any point. (All delay lines are initialized at zero.)

The above two rules are effected in a fairly simple manner. A binary storage cell is assigned in the *compiler* program for each of the output cells in the *object* program. The two states of each of these cells are called "full" and "empty." Initially all cells which represent inputs to delay lines are marked "full" and all others marked "empty." A box can be compiled whenever its inputs are all marked "full" and its output "empty." When a box is compiled, its output is marked "full" and its inputs "empty." Compilation proceeds until all boxes have been compiled (successful compilation) or until no uncompiled box meets the two requirements. In the latter event the compiler prints the remark CLOSED LOOP WITH NO DELAYS OR ALL DELAYS and halts. To see that one of these conditions must prevail, note that a delay line can *always* be compiled unless it feeds another uncompiled delay line. Therefore, if any of the uncompiled boxes are delay lines there exists a closed loop with all delays. If, on the other hand, all uncompiled boxes are nondelaying, then each must have an empty input which must be the output of an uncompiled nondelaying box. Thus, working backwards, we find a closed loop with no delay.

The order of searching for uncompiled boxes which meet the tests is immaterial from a logical point of view, but this freedom can be used to optimize the use of the accumulator. By first trying to compile a box which is fed by the last compiled box, the compiler is sometimes able to save a "storage" or "fetch" order, or both. Delays are not, of course, limited to unit delays as in the above discussion. A delay of length n sets $n - 1$ storage cells in addition to its normal output cell. For short delays the data are "stepped along" a notch each time the main loop is executed. For longer delays the same effect is produced by address modification.

The above description is merely a sketch of the general procedure used by BLODI. Actually, it has a lot more structure of purely technical character. For example, some boxes are broken down into simpler boxes by the compiler so that parts of the object program concerning a given box may not appear in consecutive locations.

V. CONCLUSIONS

BLODI has been in use at Bell Telephone Laboratories for about a year, mostly in the Department of Visual and Acoustics Research. It has been used chiefly for signal processing types of problems, and one of the authors has used the compiler to simulate a speech synthesizer of the resonant-vocoder type. It has also been used to study television coding schemes, artificial reverberation in acoustic research, and for part of the coding in a handwriting recognition problem.[1] In appraising its value one must consider two separate questions:

1. What type of problem is easily coded in the BLODI language?
2. What type of problem causes BLODI to write efficient object programs?

The first question will be answered relative to a programmer well versed in basic and Fortran language. Whenever the problem involves a rather smooth flow of data in and out of the machine, with the output being a nearly stationary function of the input, then it can be more easily coded in BLODI than in any other existing language. When, however, the program must process input samples in a complicated order, dependent on previous results, the BLODI language becomes unbearably awkward. (This is precisely the type of circuit which is hard to design with delay lines, switches, etc.)

The second question is easily answered. Any diagram which contains many idle boxes will be inefficient, because the object program goes through the motion of calculating the state of all boxes at each clock time. For example, a program with five memory-free (delay-free) paths, only one of which is connected to the output at any one time, would result in an inefficient object program. For diagrams containing few idle boxes, however, BLODI produces object programs which are usually as efficient as those written by a competent programmer.

The version of BLODI in use at Bell Telephone Laboratories is coupled to the monitor and I-O system, BE SYS 3. Thus an installation not us-

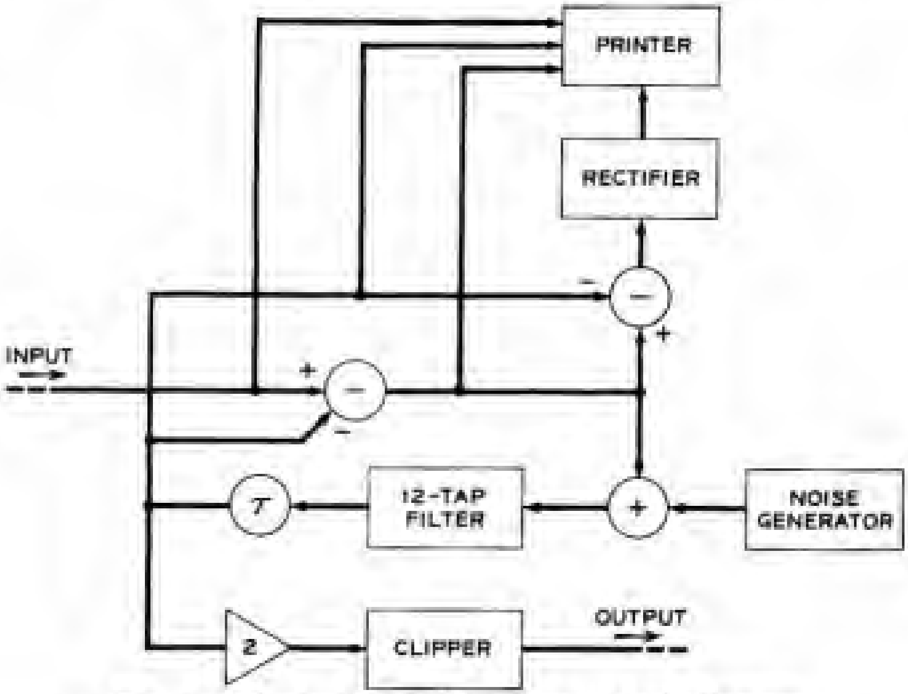

Fig. 1 — Typical BLODI program: block diagram.

```
                    SAMPLE BLODI SOURCE PROGRAM

        WNG     SUM/1,100
SUM     ADR     BUFF
BUFF    FLT     12,1,.001,.002,.004,.008,.016,.032
                .063,1/8,.230,1/4,1/5,.057,DELAY
DELAY   DEL     1,SCALE,SUB/2
                PRINT/3,SUB1/2         XXX
SCALE   AMP     2,CLIP
CLIP    CLM     T2
T2      OUT     5
                ,,,1,1                 XXX
SUB     SUB     SUM/2
                SUB1/1,PRINT/4         XXX
T1      INP     ,,,1,SUB/1
                PRINT/2,1,1            XXX

SUB1    SUB     R                      THESE CARDS AND THE CARDS MARKED
R       FWR     PRINT/1                XXX WOULD BE OMITTED IF
PRINT   PRT     150                    PRINTING WERE NOT DESIRED

        END
```

Fig. 2 — Typical BLODI program: source program.

ing BE SYS 3 would have to modify the BLODI program. The changes, however, would only involve I-O and interaction with the FAP (or SAP) assembly program.

Figs. 1, 2, and 3 represent a sample BLODI program. Fig. 1 is a block diagram suitable for simulation using BLODI; Fig. 2 is the corresponding source program; and Fig. 3 shows the printed output that results from compiling the source program and running the simulation.

REFERENCE

1. Frishkopf, L. S., and Harmon, L. D., Machine Recognition of Cursive Script, Fourth Annual Symposium on Information Theory, 1960.

```
PAGE 1
                      02734         ORG     1500
                      00101  READ   EQU  65
                      00215  WRITE  EQU     141
                      00401  MRN    EQU     257
                      00417  SRI    EQU     271
                      00517  CRS    EQU     335
      02734  0 00000 0 00001  C1    PZE 1
      02735  0540 00 0 02736  B     TSX *+1
      02736  0760 00 0 00010        RCT
      02737  0761 00 0 00000        NOP
      02740  2 00000 0 02741        TXH A
      02741  0500 00 0 02740        CLA *-1
      02742  0601 00 0 02740        STO *-2
      02743  0534 00 1 02757        LXA L,1
      02744  0634 00 1 02757        SXA L,1
      02745  2 00001 1 02744        TIX *-1,1,1
      02746  0534 00 1 02266  A     LXA COMB,1
      02747  0500 00 1 00000        CLA D+02000,1                 T1
      02750  0601 00 0 03317        STO T+1
      02751  0500 00 0 03317        CLA T+2                    DELAY
      02752  0601 00 0 03320        STO T+3
      02753  0767 00 0 00001        ALS 01                     SINLE
      02754  0560 00 0 03266        LDQ C                       CLIP
      02755  0040 00 0 02757        TLQ *+2
      02756  0500 00 0 03266        CLA C
      02757  0601 00 1 03777        STO D+57777,1                 T2
      02760  0500 00 0 03321        CLA T+4                     BUFF
      02761  0765 00 0 00043        LRS 43
      02762  0200 00 0 03267        MPY C+1
      02763  0767 00 0 00021        ALS 21    00
      02764  0760 00 0 00010        RND
      02765  0601 00 0 03324        STO T+15
      02766  0500 00 0 03325        CLA T+16
      02767  0601 00 0 03323        STO T+14
      02770  0765 00 0 00043        LRS 43
      02771  0200 00 0 03270        MPY C+2
      02772  0767 00 0 00021        ALS 21    00
      02773  0760 00 0 00010        RND
      02774  0601 00 0 03336        STO T+27
      02775  0500 00 0 03337        CLA T+28
      02776  0601 00 0 03335        STO T+26
      02777  0765 00 0 00043        LRS 43
      03000  0200 00 0 03271        MPY C+3
      03001  0767 00 0 00021        ALS 21    00
      03002  0760 00 0 00010        RND
      03003  0601 00 0 03330        STO T+21
      03004  0500 00 0 03331        CLA T+22
      03005  0601 00 0 03327        STO T+20
      03006  0765 00 0 00043        LRS 43
      03007  0200 00 0 03272        MPY C+4
      03010  0767 00 0 00021        ALS 21    00
      03011  0760 00 0 00010        RND
      03012  0601 00 0 03332        STO T+23
      03013  0500 00 0 03333        CLA T+24
      03014  0601 00 0 03331        STO T+22
      03015  0765 00 0 00043        LRS 43
      03016  0200 00 0 03273        MPY C+5
      03017  0767 00 0 00021        ALS 21    00
```

```
PAGE 2
      03020  0760 00 0 00010        RND
      03021  0601 00 0 03334        STO T+25
      03022  0500 00 0 03335        CLA T+26
      03023  0601 00 0 03333        STO T+24
      03024  0765 00 0 00043        LRS 43
      03025  0200 00 0 03274        MPY C+6
      03026  0767 00 0 00021        ALS 21    00
      03027  0760 00 0 00010        RND
      03030  0601 00 0 03337        STO T+28
      03031  0500 00 0 03337        CLA T+30
      03032  0601 00 0 03336        STO T+29
      03033  0765 00 0 00043        LRS 43
      03034  0200 00 0 03275        MPY C+7
      03035  0767 00 0 00021        ALS 21    00
      03036  0760 00 0 00010        RND
      03037  0601 00 0 03340        STO T+31
      03040  0500 00 0 03341        CLA T+32
      03041  0601 00 0 03337        STO T+30
      03042  0767 00 0 00021        ALS 21    00
      03043  0760 00 0 00010        RND
      03044  0601 00 0 03342        STO T+33
      03045  0500 00 0 03343        CLA T+34
      03046  0601 00 0 03341        STO T+32
      03047  0765 00 0 00043        LRS 43
      03050  0200 00 0 03276        MPY C+10
      03051  0767 00 0 00021        ALS 21    00
      03052  0760 00 0 00010        RND
      03053  0601 00 0 03344        STO T+35
      03054  0500 00 0 03345        CLA T+36
      03055  0601 00 0 03343        STO T+34
      03056  0767 00 0 00021        ALS 21    00
      03057  0760 00 0 00010        RND
      03060  0601 00 0 03346        STO T+37
      03061  0500 00 0 03347        CLA T+40
      03062  0601 00 0 03345        STO T+36
      03063  0765 00 0 00043        LRS 43
      03064  0200 00 0 03277        MPY C+11
      03065  0767 00 0 00021        ALS 21    00
      03066  0760 00 0 00010        RND
      03067  0400 00 0 03340        ADD T+7
      03070  0400 00 0 03344        ADD T+35
      03071  0400 00 0 03342        ADD T+33
      03072  0400 00 0 03330        ADD T+21
      03073  0400 00 0 03336        ADD T+27
      03074  0400 00 0 03324        ADD T+15
      03075  0400 00 0 03332        ADD T+23
      03076  0400 00 0 03330        ADD T+21
      03077  0400 00 0 03326        ADD T+17
      03100  0400 00 0 03324        ADD T+15
      03101  0601 00 0 03350        STO T+41
      03102  0500 00 0 03317        CLA T+10                    SUBL
      03103  0402 00 0 03312        SUB T+3
      03104  0601 00 0 03316        STO T+7
      03105  0402 00 0 03312        SUB T+3                     SHFT
      03106  0760 00 0 00002        CLM                            &
      03107  0560 00 0 03300        LDQ C+12                   FFINT
      03110  1040 00 0 03312        CLS *+2
      03111  0020 00 0 02537        TRA A+37
```

```
PAGE 3
      03112  0500 00 0 03301        CLA C+13
      03113  0100 00 0 03153        TZE *+41
      03114  0402 00 0 02734        SUB C1
      03115  0601 00 0 03301        STO C+13
      03116  0754 00 1 00000        PXD 0,1
      03117  0771 00 0 00022        ARS 22
      03120  0402 00 0 03347        SUB T+27
      03121  0602 00 0 03302        SLW C+14
      03122  0500 00 0 03303        CLA C+15
      03123  0400 00 0 02734        ADD C1
      03124  0601 00 0 03303        STO C+15
      03125  0074 00 4 77714        TSX (TPHC),4
      03126  0 77777 0 03142            [illegible],,-1
      03127  0560 00 0 03303        LDQ C+15
      03130 -1 00000 0 00000        STR
      03131  0560 00 0 03302        LDQ C+14
      03132 -1 00000 0 00000        STR
      03133  0560 00 0 03317        LDQ T+10
      03134 -1 00000 0 00000        STR
      03135  0560 00 0 03312        LDQ T+3
      03136 -1 00000 0 00000        STR
      03137  0560 00 0 03310        LDQ T+1
      03140 -1 00000 0 00000        STR
      03141  0074 00 4 77024        TSX (FIL),4
      03142  0020 00 0 03153        TRA *+11
      03143  746620604731           BCD 2COH PRINTSHIFT
      03144  214562604560
      03145  053460606060
      03146 -3 77777 3 77777        SVN -1,7,-1
      03147  0 00000 0 00766            2+1
      03150  0500 00 0 03302        CLA C+14
      03151  0400 00 0 02734        ADD C1
      03152  0601 00 0 03302        STO C+14
      03153  0074 00 4 00401        TSX MRL,4
      03154  0765 00 0 00043        LRS 43
      03155  0200 00 0 03304        MPY C+16
      03156  0767 00 0 00021        ALS 21    00
      03157  0760 00 0 00010        RND
      03160  0400 00 0 03347        ADD T+40                     SUM
      03161  0601 00 0 03347        STO T+40                    BUFF
      03162  0765 00 0 00043        LRS 43                      BUFF
      03163  0200 00 0 03305        MPY C+17
      03164  0767 00 0 00021        ALS 21    00
      03165  0760 00 0 00010        RND
      03166  0400 00 0 03350        ADD T+41                    BUFF
      03167  0601 00 0 03311        STO T+2
      03170  2 00001 4 03142        TIX A+2,4,1
      03171 -0140 00 0 03201        TNO *+10
      03172  0074 00 4 77004        TSX (PRINT),4
      03173  0 00004 0 03177            *+2,,4
      03174  0020 00 0 03201        TRA *+5
      03175  602325642564           BCD 4OUTPUT OVERFLOWED
      03176  216626514366
      03177  514346622425
      03200  246060606060
      03201  0074 00 4 00215        TSX WRITE,4
      03202  0 00000 0 03750            [illegible],,00766
      03203  0 00000 0 00005            0+5
```

```
PAGE 4
      03204  0534 00 4 03300        LXA C+00020,4
      03205 -2 00000 4 03233        TNX *+16,4,
      03206  3 00001 4 03207        TXH *+1,4,1
      03207  0074 00 4 77714        TSX (TPH),4
      03210  0 77777 0 03217            *+7,,-1
      03211  0534 00 2 03265        LXA SIZE,2
      03212  0560 00 2 07777        LDQ D+57777,2
      03213 -1 00000 0 00000        STR
      03214  2 00001 2 03212        TIX *-2,2,1
      03215  0074 00 4 77004        TSX (FIL),4
      03216  0020 00 0 03223        TRA *+6
      03217  743120606060           BCD 2COH         T2(PHG)
      03220  606060606060
      03221  011045602460
      03222 -3 77777 7 77777        SVN -1,7,-1
      03223 -0534 00 4 03300        LXD C+00020,4
      03224  1 00001 4 03226        TXI *+2,4,1
      03225 -0634 00 4 03300        SXD C+00020,4
      03226  3 00001 4 03263        TXH EXIT,4,1
      03227  0000 00 0 03352     A  STO T+00043
      03230  0074 00 4 00401        TSX MRN,4
      03231  0 00100 0 64000            [illegible]
      03232  0 00000 0 00005            0+5
      03233 -0534 00 4 03300        LXD C+00020,4
      03234 -2 00000 4 03252        TNX *+16,4,1
      03235  3 00001 4 03252        TXH *+15,4,1
      03236  0074 00 4 77714        TSX (TPH),4
      03237  0 77777 0 03246            *+7,,-1
      03240  0534 00 2 03265        LXA SIZE,2
      03241  0560 00 2 03777        LDQ D+40000,2
      03242 -1 00000 0 00000        STR
      03243  2 00001 2 03241        TIX *-2,2,1
      03244  0074 00 4 77024        TSX (FIL),4
      03245  0020 00 0 03252        TRA *+5
      03246  743144606060           BCD 2COH         T1(PH)
      03247  606060606060
      03250  014504502460
      03251 -3 77777 7 77777        SVN -1,7,-1
      03252  0500 00 0 00100        CLA READ
      03253  0100 00 0 03255        TZE *+2
      03254  0600 00 0 00100        STZ READ-1
      03255  0770 00 0 03215        WEFF     +5
      03256  0500 00 0 02734        CLA C1
      03257  0534 00 4 03352        LXA T+00043,4
      03260  0601 00 0 03350        STO T+00042
      03261 -3 00100 4 03230        TXL A+1,4,6
      03262  0020 00 0 02546        TRA EXIT+1
      03263  0770 00 0 03263  EXIT  WEFF     +9
      03264  0020 00 0 77001  EXIT1 TRA SYSTEM
      03265  0 00000 0 00765  SIZE      0+765
                      00766  D      BSS 0+765
      03266 +000000000200     C     OCT +000000000200
      03267 +000000000407           OCT +000000000407
      03270 +000000001014           OCT +000000001014
      03271 +000000002021           OCT +000000002021
      03272 +000000004061           OCT +000000004061
      03273 +000000010141           OCT +000000010141
      03274 +000000020305           OCT +000000020305
```

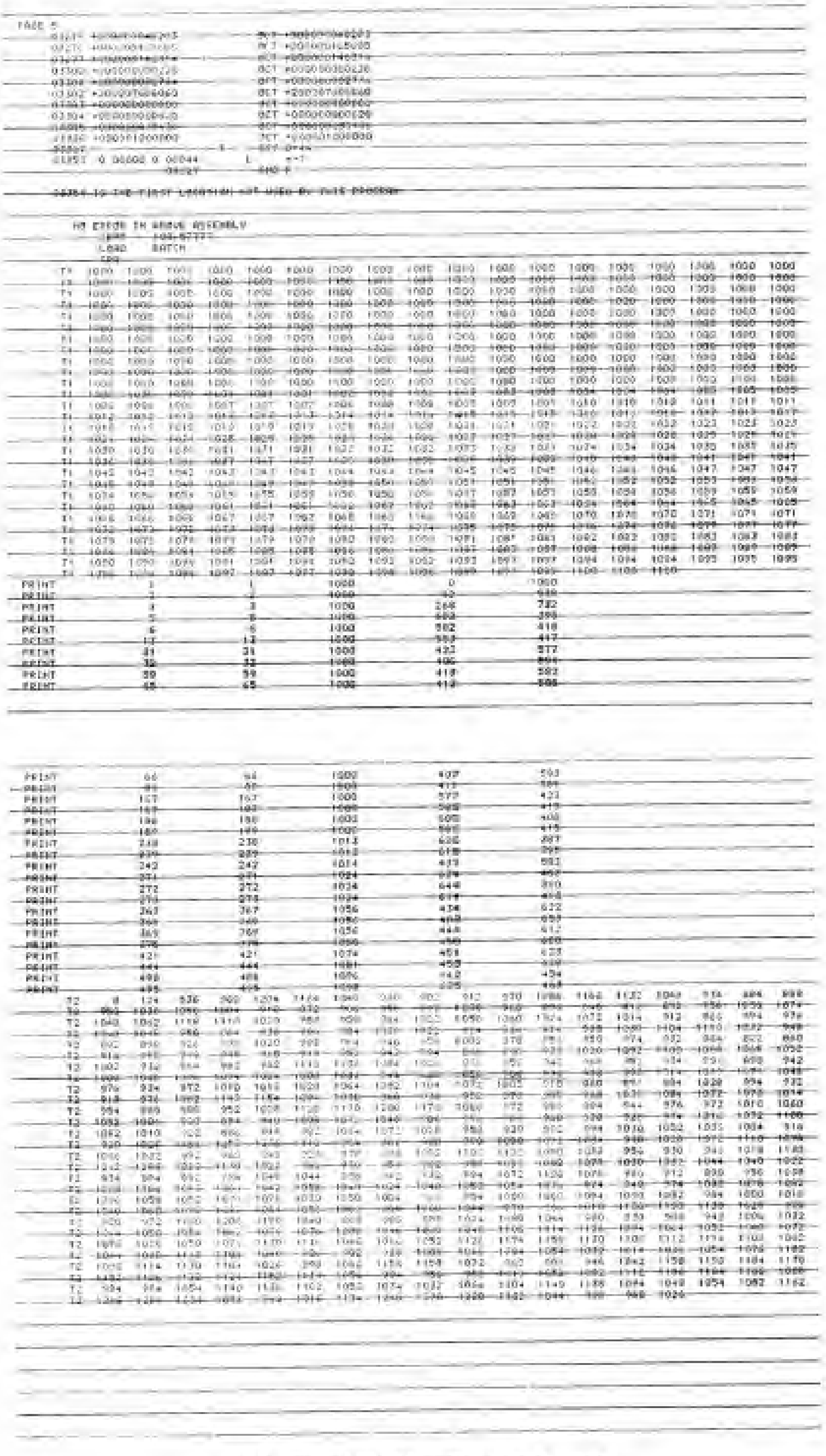

Fig. 3 — Typical BLODI program: printed output.